VisionSeed SDK 文档

腾讯科技（上海）有限公司

版本历史

日期	版本	发布说明
2019/06/24	1.0	初始版本
2019/08/19	1.1	增加人脸识别相关接口

目录

1. 概述

VisionSeed SDK 是 VisionSeed 的软件开发包，提供外部设备与 VisionSeed 通信的 API 接口。本文档主要介绍了 VisionSeed SDK 提供的 API 接口以及 SDK 的软件结构。

VisionSeed SDK 提供整套源代码，您可以根据选用的不同平台自行编译源代码和示例程序。从开发者中心-工具-VisionSeed 下载 SDK 源代码，目录结构如下：

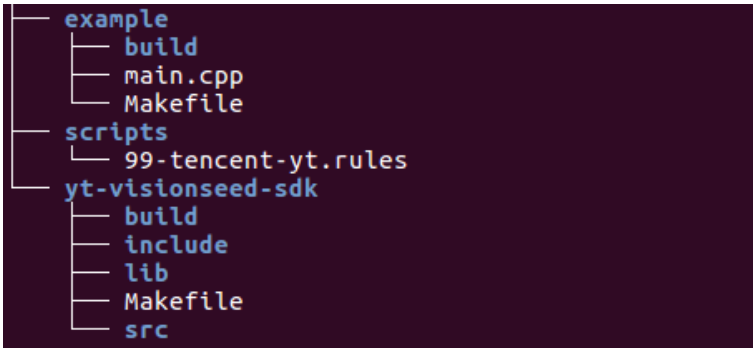

图 1.1 SDK 目录结构

其中 **scripts/99-tencent-yt.rules** 为 USB 权限配置文件，可授予普通用户 USB 接口权限，需要执行以下命令将该文件复制到文件夹**/etc/udev/rules.d/**。

```
sudo cp ./scripts/99-tencent-yt.rules /etc/udev/rules.d/

udevadm trigger
```

yt-visionseed-sdk 文件夹包含 sdk 源码，通过运行 make 编译源

```
cd yt-visionseed-sdk/
make
cd ..
```

example 文件夹包含示例程序，通过运行 make 编译程序

```
cd example/
make
```

以下章节将对 SDK API 以及 SDK 结构进行说明。

2. API

VisionSeed SDK 提供了简单的接口类 YtVisionSeed，所有的 API 均通过该类导出，如下为该类提供的函数列表。

图 2.1 API 导出类

通过提供一个串口设备地址，可以实例化一个 YtVisionSeed，然后通过 visionseed 调用 API 接口即可。如下为示例，其中**/dev/ttyACM0** 为通过 USB 连接 VisionSeed 到 PC 之后的设备文件（请使用实际的设备文件名）。

```
YtVisionSeed visionseed("/dev/ttyACM0"); // 实例化 YtVisionSeed
```

实例化 YtVisionSeed 之后即可通过 visionseed 调用 API 接口，以下分类介绍 API 函数。

2.1 设备

获取设备信息

```
std::string YtVisionSeed::GetDeviceInfo();
```

返回值：

以空格分隔的字符串，格式为**"vx.x.x_compileID CPUID MODULEID"**，其中 vx.x.x_ compileID 为固件版本号，CPUID 为 CPU 的序列号，MODULEID 为摄像头模组的序列号。

2.2 相机

设置自动曝光

```
bool YtVisionSeed::SetCamAutoExposure(CameraID camId);
```

参数：

camId：CAMERA_RGB 或 CAMERA_IR

返回值：

true　设置自动曝光成功

false　设置自动曝光失败

设置手动曝光

```
bool YtVisionSeed::SetCamManualExposure(CameraID camId,int32_t
timeUs,int32_t gain);
```

参数：

camId：CAMERA_RGB 或 CAMERA_IR

timeUs：1~30000

gain：0~100

返回值：

true 设置手动曝光成功

false 设置手动曝光失败

设置IR补光灯的亮度值

```
bool YtVisionSeed::SetFlasher(int32_t flasherIR, int32_t flasherWhite);
```

参数：

flasherIR：0~100

flasherWhite：保留参数，未使用

返回值：

true 设置亮度值成功

false 设置亮度值失败

2.3 人脸库管理

获取已入库人脸列表

```
std::vector<FaceIdInfo> YtVisionSeed::ListFaceId();
```

参数：

无

返回值：

FaceIdInfo 定义如下：

```
typedef struct _FaceIdInfo {
```

```
    int32_t faceId;
    char faceName[32];
} FaceIdInfo;
```

上传照片进行入库

```
YtRpcResponse_ReturnCode YtVisionSeed::RegisterFaceIdWithPic(std::string
path, std::string faceName, int32_t *faceIdOut);
```

参数：

path：本地文件路径，支持 jpg 格式，1280x720 分辨率以下，可以是完整图片，也可以是人脸区域小图。

faceName：名称，最大 31 字符+1 个 NULL 结束符。

faceId：如果不为 NULL，则返回注册成功的 faceId。

返回值：

YtRpcResponse_ReturnCode_SUCC：成功
YtRpcResponse_ReturnCode_ERROR_REGISTER_FACEID_NO_FACE_DETECTED：未检测到人脸
YtRpcResponse_ReturnCode_ERROR_REGISTER_FACEID_TOO_MANY_FACES：人脸太多
YtRpcResponse_ReturnCode_ERROR_REGISTER_FACEID_FILE_NOT_READABLE：文件异常
YtRpcResponse_ReturnCode_ERROR_REGISTER_FACEID_LIB_FULL：库满了

从摄像头拍摄画面进行入库

```
YtRpcResponse_ReturnCode YtVisionSeed::RegisterFaceIdFromCamera(std::string

faceName, int32_t timeoutMs, int32_t *faceIdOut);
```

参数：

faceName：名称，最大 31 字符+1 个 NULL 结束符。

timeoutMs：超时时间（毫秒）。

faceId：如果不为 NULL，则返回注册成功的 faceId。

返回值：

YtRpcResponse_ReturnCode_SUCC：成功
YtRpcResponse_ReturnCode_ERROR_REGISTER_FACEID_NO_FACE_DETECTED：未检测到人脸
YtRpcResponse_ReturnCode_ERROR_REGISTER_FACEID_TOO_MANY_FACES：人脸太多
YtRpcResponse_ReturnCode_ERROR_REGISTER_FACEID_LIB_FULL：库满了
YtRpcResponse_ReturnCode_ERROR_REGISTER_FACEID_TIMEOUT：指定时间内没有检测到合格人脸

更改库中 FaceId 的名称

```
bool YtVisionSeed::SetFaceName(int32_t faceId, std::string faceName);
```

参数：

faceId：要操作的 id

faceName：名称，最大 31 字符+1 个 NULL 结束符。

返回值：

true 成功

false 失败

删除 faceName 匹配的条目

```
int32_t YtVisionSeed::DeleteFaceName(std::string faceName);
```

参数：

faceName：名称，最大 31 字符+1 个 NULL 结束符。

返回值：

成功删除的数量

删除指定条目

```
bool YtVisionSeed::DeleteFaceId(int32_t faceId);
```

参数：

faceId：要操作的 id

返回值：

true 成功

false 失败

删除所有条目

```
bool YtVisionSeed::ClearFaceLib();
```

参数：

无

返回值：

true 成功

false 失败

2.4 获取人脸检测结果

2.4.1 注册回调函数

```
void YtVisionSeed::RegisterOnFaceResult(OnResult callback);
```

注册人脸检测结果回调函数，VisionSeed 人脸检测结果将通过回调函数传回，目前人脸的检测速度约为 10fps，即在场景中有人脸的情况下，回调函数的调用频率约为 10Hz。

参数：

callback:回调函数

使用示例：

```
// 定义回调函数
void OnFaceRetrieveResult(shared_ptr<YtMsg> ytmsg)
{
    // 在这里通过解析 YtMsg 得到人脸检测的结果
}
// 实例化 YtVisionSeed
YtVisionSeed visionseed("/dev/ttyACM0");
//注册结果获取接口，注册之后会在检测到人脸信息的时候调用
OnFaceRetrieveResult 回调函数
visionseed.RegisterOnFaceResult(OnFaceRetrieveResult);
```

SDK 将 YtMsg 作为参数调用回调函数，在回调函数中通过解析 YtMsg 可以获取到人脸检测的各种结果信息，以下分类进行说明。

2.4.2 获取帧信息

```
#define VSRESULT(vs_result) (vs_result)->values.result
uint64_t frameIndex = VSRESULT(ytmsg).frameId;
uint64_t frameTimestampUs = VSRESULT(ytmsg).frameTimestampUs;
```

使用 VSRESULT 宏可以从 YtMsg 中得到当前帧的编号 frameIndex，该参数单调递增，以及当前帧的时间戳。

2.4.3 获取检测到的人脸数目

```
#define VSRESULT_DATA(vs_result) (vs_result)->values.result.data
size_t faceCnt = VSRESULT_DATA(ytmsg).faceDetectionResult.face_count;
```

使用 VSRESULT_DATA 宏可以从 YtMsg 中得到检测到的人脸数量 faceCnt。

2.4.4 获取轨迹 ID

轨迹 ID（traceId）是一个人脸进入画面后自动生成的唯一标识，他不会随着运动、人脸大小的变化而改变。

```
for (size_t i = 0; i < faceCnt; i++)
{
    if (VSRESULT_DATA(message).faceDetectionResult.face[i].has_traceId)
    {
        uint32_t traceId = VSRESULT_DATA(message).faceDetectionResult.face[i].traceId;
        printf("traceId: %u\n", traceId);
    }
}
```

2.4.5 获取 1:N 识别结果

通过以下代码遍历所有检测到的人脸，如果该人脸与库中条目匹配，会通过 name 属性输出。

```
for (size_t i = 0; i < faceCnt; i++)
{
    if (VSRESULT_DATA(message).faceDetectionResult.face[i].has_name &&
VSRESULT_DATA(message).faceDetectionResult.face[i].has_nameConfidence)
    {
        const char *name =
VSRESULT_DATA(message).faceDetectionResult.face[i].name;
        float confidence =
VSRESULT_DATA(message).faceDetectionResult.face[i].nameConfidence;
        printf("Who: %s (confidence: %0.3f)\n", name, confidence);
    }
}
```

2.4.6 获取人脸框

可以通过以下代码遍历所有检测到的人脸框信息，同一帧上最多检测出 8 个人脸框，按人脸大小排序，大的在先（face 数组索引小），小的在后（face 数组索引大）。

```
for (size_t i = 0; i < faceCnt; i++)
{
    const Rect& rect = VSRESULT_DATA(ytmsg).faceDetectionResult.face[i].rect;
    printf("faceRect, x:%d y:%d, width:%d, height:%d\n",
                rect.x, rect.y, rect.w, rect.h);
}
```

Rect 定义如下，

```
typedef struct _Rect {
    int32_t x; // 左上角 x 坐标
    int32_t y; // 左上角 y 坐标
    int32_t w; // 宽度
    int32_t h; // 高度
} Rect;
```

2.4.7 获取姿态角

VisionSeed 能够根据检测到的人脸信息计算出头部的空间姿态信息，返回 roll，pitch 和 yaw 角度信息，可以通过以下代码遍历所有检测到的人脸姿态角信息，

```
for (size_t i = 0; i < faceCnt; i++)
{
    if (VSRESULT_DATA(ytmsg).faceDetectionResult.face[i].has_pose)
    {
```

```
            const Pose& p =
VSRESULT_DATA(ytmsg).faceDetectionResult.face[i].pose;
            printf("roll:%0.3f, pitch:%0.3f, yaw:%0.3f\n", p.roll, p.pitch, p.yaw);
        }
}
```

其坐标系定义如下，

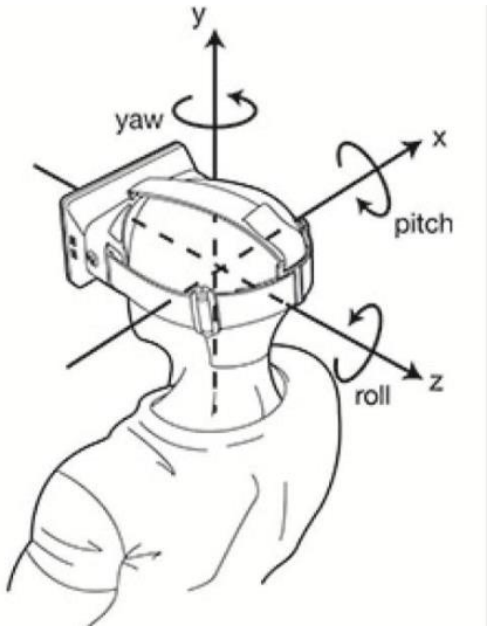

图 2.2 姿态坐标系定义

姿态结构体定义如下，

```
typedef struct _Pose {
    float roll;
    float yaw;
    float pitch;
} Pose;
```

各参数的取值范围以及准确度如下表所示，

参数名称	类型	描述
roll	**float**	脸部左右倾斜角度，取值范围-180~180，在-60-60 内效果准确，角度偏移过大可能造成检测失败或者点位不准

yaw	**float**	脸部左右扭头角度，取值范围-90~90，在-30-30 内效果准确，角度偏移过大可能造成检测失败或者点位不准
pitch	**float**	脸部俯仰角，取值范围-90~90，在-30-30 内效果准确，角度偏移过大可能造成检测失败或者点位不准

2.4.8 获取面部 90 个配准点

人脸配准可以根据人脸框的位置获取人脸五官点坐标。VisionSeed 人脸检测算法输出 90 个人脸配准点信息 YtFaceShape。可以通过以下代码获取这些配准点信息。同一帧上最多检测出 8 个人脸框，对于其中较大的 4 个脸，会计算 90 点配准点信息。

```
for (size_t i = 0; i < faceCnt; i++)
{
    YtFaceShape shape;
    if (GetYtFaceShape(&(VSRESULT_DATA(ytmsg).faceDetectionResult.face[i]), shape))
    {
        // 得到左眼的配准点信息
        for (int j = 0; j < sizeof(YT_FACE_SHAPE_SIZE_LEFT_EYE); j++)
        {
            printf("x:%0.3f, y:%0.3f\n", shape.leftEye[j].x, shape.leftEye[j].y);
        }
        // 类似的可以获取其他脸部配准点，比如眉毛，鼻子，嘴巴，瞳孔，面部轮廓
    }
}
```

YtFaceShape 结构体定义如下，按照不同的人脸部位分成了左眉毛，右眉毛，左眼，右眼，鼻子，嘴巴，人脸轮廓以及瞳孔几个部分。每部分由数量不等的配准点组成，总共 90 个配准点。对应每个部分的配准点信息，有对应的可见性

（Visibility）属性，一般认为可见性小于 0.5 即认为该点被遮挡。YtFaceShape 结构体还包括 90 个配准点的置信度（confidence），一般认为置信度大于 0.5 表示配准点结果可靠。

```
#define YT_FACE_SHAPE_SIZE_LEFT_EYEBROW 8
#define YT_FACE_SHAPE_SIZE_RIGHT_EYEBROW 8
#define YT_FACE_SHAPE_SIZE_LEFT_EYE 8
#define YT_FACE_SHAPE_SIZE_RIGHT_EYE 8
#define YT_FACE_SHAPE_SIZE_NOSE 13
#define YT_FACE_SHAPE_SIZE_MOUTH 22
#define YT_FACE_SHAPE_SIZE_PROFILE 21
#define YT_FACE_SHAPE_SIZE_PUPIL 2

#pragma pack(1)
struct YtFaceShape
{
    cv::Point2f leftEyebrow         [YT_FACE_SHAPE_SIZE_LEFT_EYEBROW]; // 左眉毛
    cv::Point2f rightEyebrow        [YT_FACE_SHAPE_SIZE_RIGHT_EYEBROW];// 右眉毛
    cv::Point2f leftEye             [YT_FACE_SHAPE_SIZE_LEFT_EYE]; // 左眼
    cv::Point2f rightEye            [YT_FACE_SHAPE_SIZE_RIGHT_EYE];// 右眼
    cv::Point2f nose                [YT_FACE_SHAPE_SIZE_NOSE]; // 鼻子
    cv::Point2f mouth               [YT_FACE_SHAPE_SIZE_MOUTH]; // 嘴巴
    cv::Point2f faceProfile         [YT_FACE_SHAPE_SIZE_PROFILE]; //人脸轮廓
    cv::Point2f pupil               [YT_FACE_SHAPE_SIZE_PUPIL]; // 瞳孔
    // 以下为对应配准点的可见性，一般可以认为<0.5 即为该点被遮挡。
    float leftEyebrowVisibility    [YT_FACE_SHAPE_SIZE_LEFT_EYEBROW];
    float rightEyebrowVisibility [YT_FACE_SHAPE_SIZE_RIGHT_EYEBROW];
    float leftEyeVisibility         [YT_FACE_SHAPE_SIZE_LEFT_EYE];
    float rightEyeVisibility        [YT_FACE_SHAPE_SIZE_RIGHT_EYE];
    float noseVisibility            [YT_FACE_SHAPE_SIZE_NOSE];
    float mouthEyebrowVisibility [YT_FACE_SHAPE_SIZE_MOUTH];
```

```
    float faceProfileVisibility    [YT_FACE_SHAPE_SIZE_PROFILE];
    float pupilVisibility          [YT_FACE_SHAPE_SIZE_PUPIL];
    // 配准点的置信度，一般>0.5 可以认为结果可靠
    float confidence;
};
#pragma pack()
```

参数名称	类型	描述
***Visibility**	**float**	五官点遮挡程度。分别对应了人脸 90 个点。一般可以认为<0.5 即为该点被遮挡。
confidence	**float**	本次结果的置信度。一般>0.5 可以认为结果可靠。

人脸各个部位的配准点顺序如下图示，每个部位（眼睛，鼻子等）使用不同的颜色标注，每个部位的点位顺序从 0 开始，分别对应 YtFaceShape 结构体中的成员变量。如蓝色点表示鼻子的点位信息，对应 YtFaceShape 结构体中的变量 nose：

```
cv::Point2f nose                [YT_FACE_SHAPE_SIZE_NOSE];
```

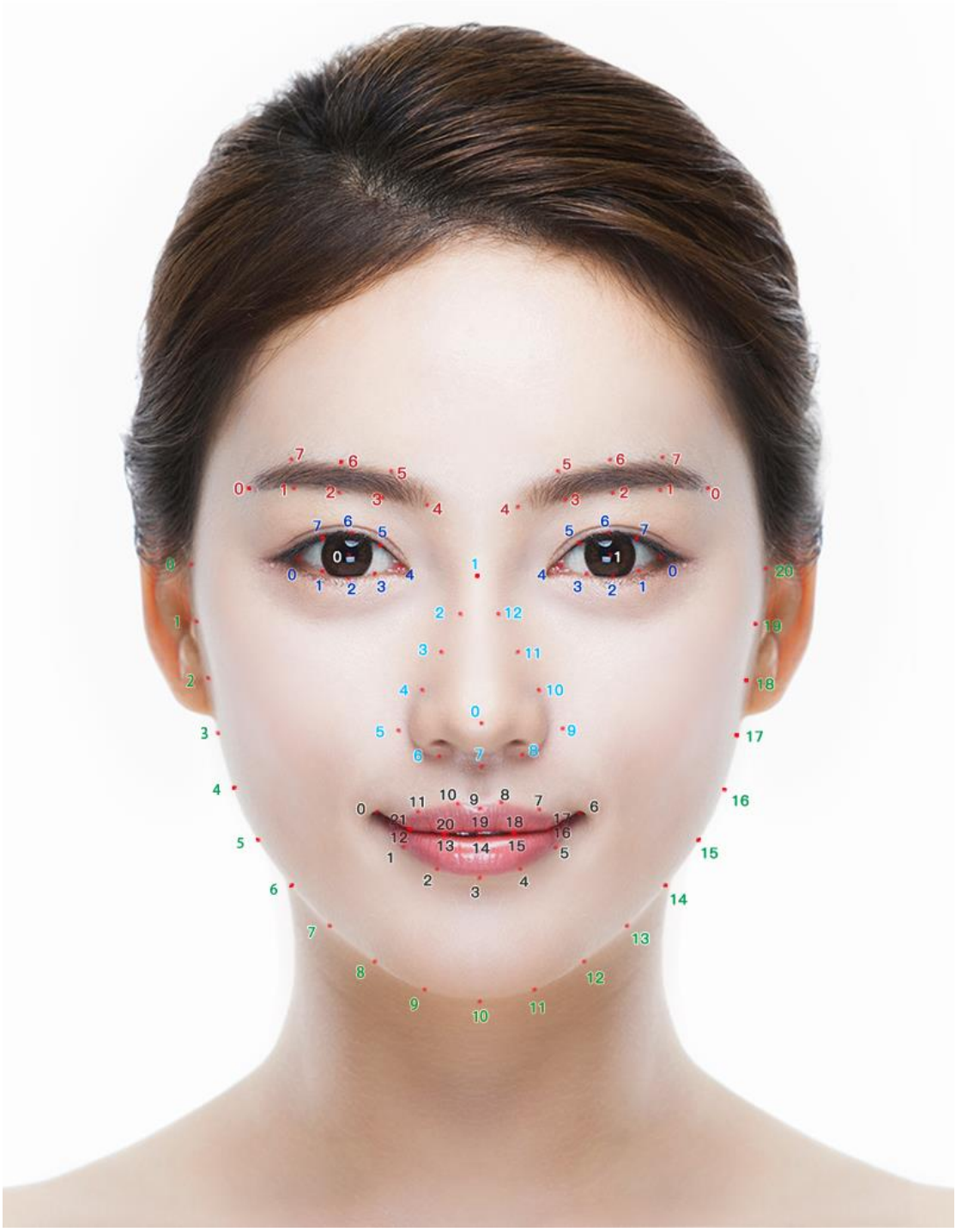

图 2.3 90 点配准信息

3. SDK 结构

本章节将以 UML 的形式介绍 SDK 的代码结构。如下图示为 SDK 的 UML 图，

YtDataLinkPushPosix 和 YtDataLinkPullPosix，分别负责数据的异步发送和接

收，YtMessenger 和 YtVisionSeed 对异步收发进行了封装，实现同步的数据收发。重要类的工作流程通过标注的形式在 UML 图中标出。

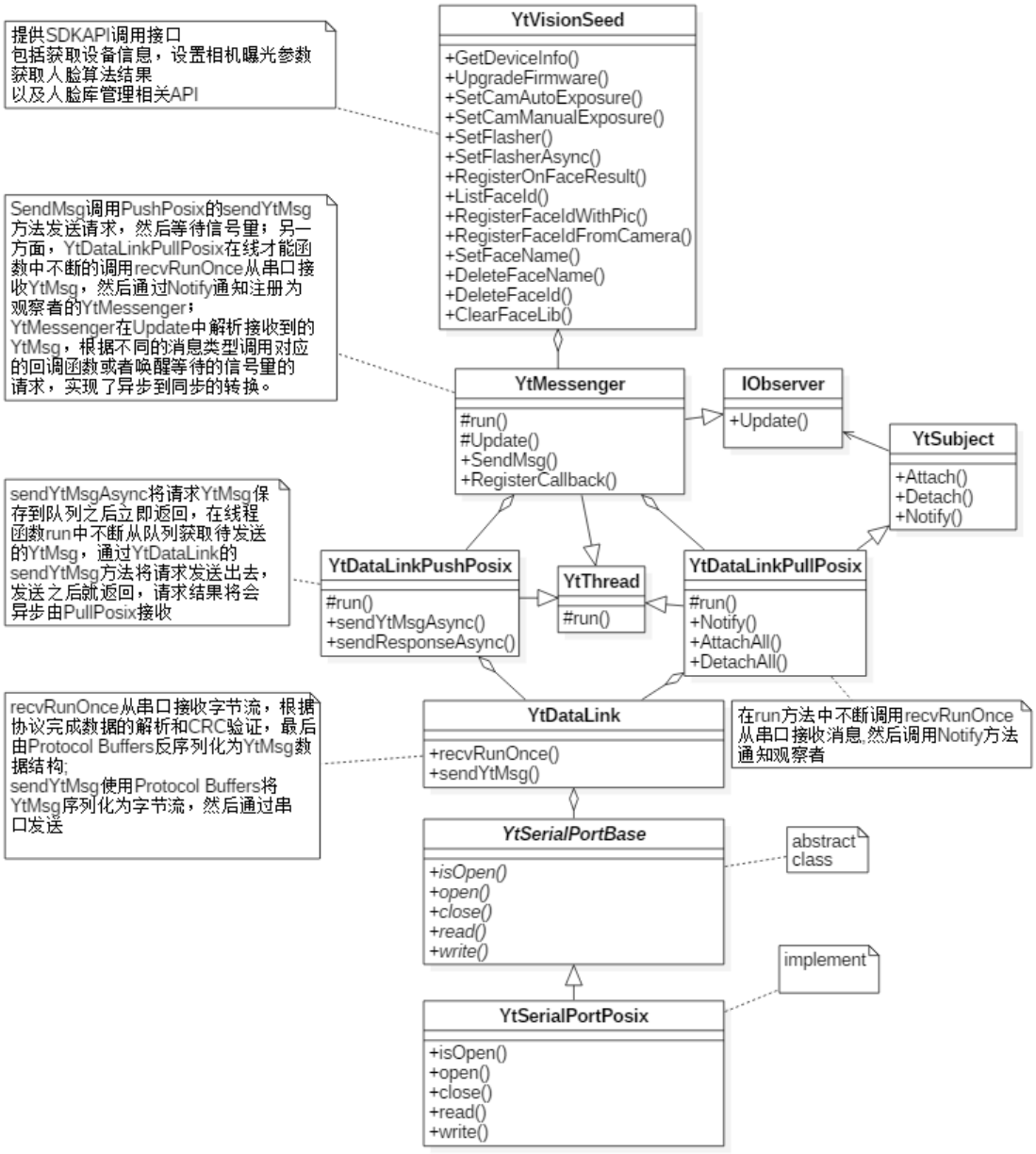

图 3.1 SDK UML

以使用 PC 与 VisionSeed 进行双向通信为例，下图描述了关键类的位置以及双向的数据流：

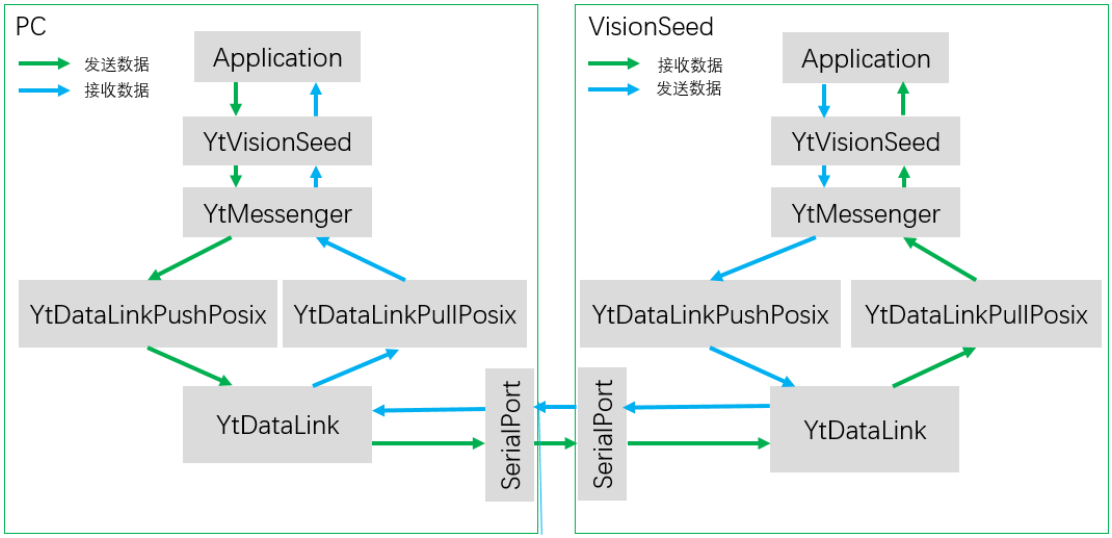
PC
发送数据
接收数据
Application
YtVisionSeed
YtMessenger
YtDataLinkPushPosix
YtDataLinkPullPosix
YtDataLink
SerialPort
VisionSeed
接收数据
发送数据
Application
YtVisionSeed
YtMessenger
YtDataLinkPushPosix
YtDataLinkPullPosix
SerialPort
YtDataLink

图 3.2 双向数据流示意